Transparent Distributed Redirection of HTTP Requests

Baggio, A.G.; van Steen, M.R.

published in
2nd International Symposium on Network Computing and Applications
2003

document version
Publisher's PDF, also known as Version of record

Transparent Distributed Redirection of HTTP Requests

Aline Baggio Maarten van Steen
Vrije Universiteit – Department of Computer Science
De Boelelaan 1081a – 1081HV Amsterdam
The Netherlands
{baggio,steen}@cs.vu.nl

Abstract

Replication in the World-Wide Web covers a wide range of techniques. Often, the redirection of a client browser towards a given replica of a Web page has to be explicit and is performed after the client's request has reached the Web server storing the requested page. As an alternative, we propose to perform the redirection as close to the client as possible in a fully distributed manner. Distributed redirection ensures that we find a replica wherever it is stored and that the closest possible replica is always found first. By exploiting locality, we can keep latency low.

1 Introduction

Replication in the World-Wide Web encompasses a wide range of techniques, from proxy caches to mirrors and Content Distribution Networks (CDNs). The use of location-dependent URLs does not facilitate transparent access to the replicated Web pages. Instead, it is often necessary to explicitly *redirect* clients towards a given replica.

Redirection in the case of proxy caches occurs in an implicit way: each HTTP request is routed through the cache or the hierarchy of caches and, in the best case, the replica of the requested Web page is retrieved directly from the cache storage space. With mirrors or CDNs, the client browser has to be explicitly redirected to a machine which is normally not on the route followed by the request. Redirection in these cases is generally achieved in a centralized way. A client is redirected only after its request has reached the *home* server, that is, the host named in the document's URL, which then decides on the replica server that should further handle the request.

The main disadvantage of centralized redirection mechanisms is the induced latency. Ideally, a client request should not go to a page's home to be redirected. The redirection should take place as soon and as close to the client as possible. We have devised a distributed redirection scheme in which the redirection decision can be taken locally at the client machine or, in the worst case, before the HTTP request leaves the client's network. In this paper, we present our design and show how it can be more or less transparently integrated with the current Web.

The paper is organized as follow. Section 2 gives an overview of the existing redirection methods for the World-Wide Web. Section 3 presents the principle of our distributed redirection scheme. Section 4 details the design of a redirection server, followed in Section 5 with aspects concerning interaction with the redirection server. Section 6 discusses the integration of the redirection mechanism in the current Web. Finally, Section 7 concludes and gives some future work directions.

2 Alternatives for redirecting clients in the Web

Redirection in today's World-Wide Web is achieved in three different ways: HTTP-based redirection, DNS-based redirection and TCP handoff.

The HyperText Transfer Protocol (HTTP) is widely used for communication between browsers, servers and proxy caches [4]. Whenever a client browser requests a Web page, it contacts the Web server named in the URL. Instead of directly sending back the page contents, the Web server can decide to redirect the client browser to another server. This redirection takes the form of another URL naming the server where a replica of the requested page can be found [2, 4]. The browser has to issue a new HTTP request to fetch the Web page at the replica site.

The Transmission Control Protocol (TCP) can also be used to redirect clients [2, 5]. In TCP, communicating parties are identified by an end point: the network address of the machine on which the party resides and the port number it uses. The data exchanged between two communicating parties are sent in portions called segments. The origin end point can be falsified when producing TCP segments.

Using this feature, a Web server hosting a replica of the requested Web page can let the requesting client believe that the segment originates from the original Web server. The client browser keeps sending its requests to the original Web server. The requests are intercepted by the original Web server's gateway which forwards the requests to the Web server holding the replica. This server then responds directly to the client (with falsified origin end points). TCP-based redirection is also known as TCP handoff.

Finally, the Domain Name System (DNS) can be used for redirection purposes [2, 7]. DNS-based redirection exploits the fact that a browser needs to resolve the domain name contained in a URL to a network address. Unless the name-to-address mapping is already cached at the client's DNS, the client's DNS request reaches the DNS server responsible for the Web server's domain (i.e., the authoritative DNS server). As a reply, the authoritative server can decide to send any appropriate network address and not only the address of the Web server designated in the URL. In particular, the DNS server can respond with the address of a Web server holding a replica of the Web page. The returned address is cached at the client's DNS. The subsequent DNS requests for this domain are therefore resolved to the replica's network address until the address is flushed from the DNS cache.

Each of these three Web redirection methods have their own characteristics that make them not entirely satisfactory. Most importantly, the three methods require that the actual redirection is done by a server close to the Web server hosting the requested page. Either it is the Web server itself, the gateway or the Web server's DNS server. As a consequence, a large number of requests travel to the server side before the redirection takes place. The worst case is HTTP-based redirection where each single request has to be redirected independently of the others. Latency is thus an important disadvantage of HTTP-based redirection [2]. However, since subsequent redirections are independent of each other, HTTP-based redirection provides a fine granularity that TCP handoff and DNS-based redirection can not achieve.

Another disadvantage of HTTP-based redirection is that it does not provide support for redirection transparency. Clients are aware of the redirection since the replica's address is passed to the client. This allows a client to cache and reuse references towards replicas, which may conflict with the redirection policy of the Web server. On the other hand, TCP handoff is fully transparent but not scalable. The traffic generated by the segment forwarding makes the method more suited for long-lived sessions such as FTP [2] or for use in local-area networks. In that respect, DNS-based redirection is more scalable. A more detailed comparison of redirection mechanisms can be found in [2, 7, 8].

Caches provide an alternative for redirection. Each HTTP request a client generates may first travel to the local cache to check whether the page has already been downloaded. Hierarchical caches are based on the same principle [3]. The caches are organized as a tree. If a Web page was not found in the local cache, the request is forwarded to the upper-level cache. Hierarchical caches are commonly used. However, they guarantee neither that a request returns the closest replica, nor that a replica is inevitably found if there is one somewhere in the cache hierarchy.

Cooperative caches aim at short-cutting the forwarding of requests and at taking benefit of pages cached at low-level caches [9]. Cached data is partitioned and sometimes replicated among the different sites participating in the cooperative cache. By maintaining a directory of cache entries at each site, a cached Web page can be found in at most two hops: one hop to the local cache and to fetch the actual location of the cached page, one hop to the cache where the page is currently stored. However, two problems arise with cooperative caches [12]. First, the maintenance of the directory of cache entries is often costly. Second, the cache hit percentage gained through cooperation appears to be low in current Web, making it not worth it to maintain the directory.

3 Principles of distributed redirection

Considering the disadvantages of HTTP-based, DNS-based redirection and TCP handoff, we would like to devise a redirection method offering a fine granularity in redirection without loss of scalability or transparency. We consider scalability by locality important: a request to look for a replica of a Web page has to avoid traveling a long distance. Also, the selected replica should remain the nearest possible to the client browser.

Consider the following scenario. A Web page is referred to as http://www.cs.vu.nl/globe/ and is available at four Web servers: Amsterdam, the "home" location of the document; Naples; San Francisco and Sydney. A client browser located in San Diego issues an HTTP request for the page. This request should reach a redirection server so that the client can find a replica of the Web page. With the current redirection mechanisms, the request travels, in principle, to the home server in Amsterdam and only there is it redirected. We propose to improve locality for client HTTP requests by using a collection of redirection servers that are installed close to the clients. In our example, the browser's HTTP request is processed first by its local redirection server in San Diego (see Figure 1).

In our approach, a redirection server knows only about pages that are available in its own area. Since the requested Web page is not replicated locally in San Diego, the redirection server in San Diego itself has to issue a lookup request to find the page. In order to keep the communication costs

relatively low and to preserve locality, a redirection server always tries to find a requested Web page in its neighboring areas. This is achieved by organizing the collection of redirection servers hierarchically. Such an organization is done on a per-page basis. For each page, a separate hierarchical organization of servers assists in redirecting HTTP requests. How such an organization can be realized efficiently is discussed below. Figure 1 shows the hierarchy for the page http://www.cs.vu.nl/globe/. The page has one redirection server for each area covered by the hierarchy. This hierarchy is organized around the Amsterdam redirection server which is the home location of the Web page. It is best if the root redirection server is co-located with the home Web server. Note that the hierarchy of redirection servers of the page http://www.cs.vu.nl/globe/ can be shared with other pages whose home Web server is in Amsterdam.

The lookup request issued by the redirection server in San Diego is further treated as follows. It reaches the page's redirection server for USA. For preserving locality, only leaf servers store records with addresses of replicas. The intermediate US server thus does not have an address for the Web page. However, it holds a pointer to the redirection server in San Francisco, which is known to have information about a replica of the page. The US server further forwards the lookup request to the redirection server in San Francisco. San Francisco replies with the address of the Web page completing the lookup request. The client's redirection server in San Diego is in charge of actually retrieving the Web page from the San Francisco Web server. The San Diego server can finally decide to cache the address for handling subsequent client requests, for example, when downloading the inline images of a document.

By contacting its local redirection server, a client browser implicitly initiates a lookup for a replica at the lowest level of the redirection service. In the best case, the address of a replica can be found at this server (local replica or cached address). If not, the forwarding of the lookup request takes place. Each step up in the hierarchy broadens the search. Having replica lookups always starting locally at the client site and gradually expanding the search area guarantees that the potential local and close-by replicas are found first. This also guarantees us to find a replica wherever it is stored. The forwarding of the requests along the hierarchy allows us to avoid unnecessary communication with parties that are far apart. In addition, by keeping the number of levels in the hierarchy relatively small, we can ensure that we do not introduce too much latency when forwarding the requests.

The placement of the redirection servers is also important. A single host can run several servers acting at different levels of the hierarchy. For example, in Figure 1, the root *World* of the Web page's hierarchy is placed in Amsterdam. It would be counter-productive to have a lookup request first reaching the root in Amsterdam, having it forwarded to the European level, for example in Germany, to the Netherlands level in Rotterdam and back to Amsterdam where the leaf server is located. It is more efficient to let the set of servers {*World, Europe, The Netherlands, Amsterdam*} run at one and same host in Amsterdam. The servers running on the Amsterdam redirection host can be efficiently implemented using a single multi-threaded redirection server.

4 Detailed design

The redirection service relies on two main components: a hierarchical collection of redirection servers and the mechanisms of the redirection server itself. This section details what the hierarchy of redirection servers is and how it is built. It also describes how a redirection server works and how it makes use of the hierarchy.

4.1 A hierarchy of redirection servers

The collection of redirection servers is distributed world wide. It is organized hierarchically in such a way that each part of the globe is taken care of by one or more redirection servers. The redirection servers are themselves organized on a world-wide collection of *redirection hosts*. As it is the case for Web servers, a given host can run several redirection servers. A hierarchy of redirection servers is constructed as follows. Each server is responsible for a one and only one given *domain* and thus operates at one given level of the hierarchy. Leaf domains are aggregated into larger domains and, in turn, these domains are aggregated so that eventually the highest-level domain covers the entire network. In our example, a domain corresponds to a geographical region. The domain thus represents a notion of locality. Each domain is allocated at least one redirection server. However, there will generally be multiple redirection servers and hosts per domain. For example, the root domain will have thousands of redirection servers distributed all over the world. For each Web page, a given server of the collection will act as root server and pages hosted in different leaf domains will generally have different root servers, as suggested in [10]. The full redirection service is therefore organized as a collection of trees of redirection servers rather than as a unique hierarchy. Note that this full distribution takes care of balancing the load across *all* the servers of the redirection service. The hierarchy of domains, however, is unique.

For locality purposes, a server stores information only over the replicas that belong to its own domain. The address of a given replica is therefore to be found at *one* redirection server. Moreover, addresses are stored only at leaf servers. This makes it unnecessary to maintain consistency within

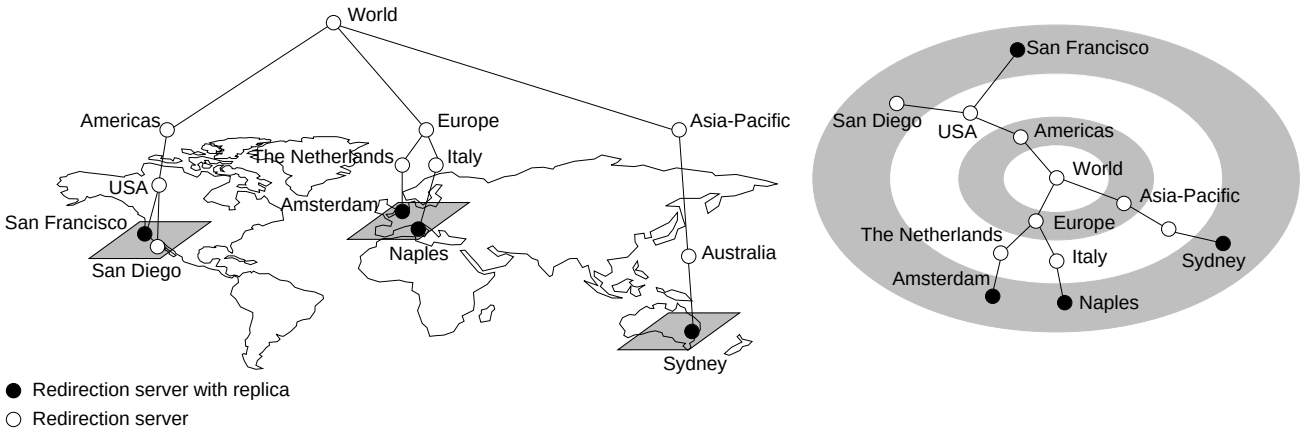

Figure 1. Building a hierarchy of redirection servers

the redirection service. As a whole, the world-wide collection of redirection servers stores the addresses (URLs) of all the replicas of the Web pages willing to participate in the service.

To find a given address starting from any redirection server in the hierarchy, a redirection server can store not only addresses of Web pages but also *forwarding pointers* to other redirection servers in one of its subdomains. Following a forwarding pointer guarantees that an address for a given replica will be found and that this replica lies in a subdomain of the redirection server where the forwarding pointer was found. In our example, the US server holds a pointer to a server in the San Francisco subdomain.

The hierarchy of domains of the redirection service reflects geographical locality. However, the locality metrics can also be expressed in terms of network distance such as latency. From now on, we assume that geographical and network distances are equivalent. Although this assumption is generally false, it has an impact only on the way the hierarchy is built. It does not influence the locality in the treatment of requests, nor does it change the way we manage the hierarchy of domains or the redirection servers. Choosing another locality metric would thus only lead to building the redirection service hierarchy in a different way. Moreover, the hierarchy can be adapted a posteriori. For each Web page, the redirection service is brought up with an initial configuration for the hierarchy of domains which is a rough estimate of what the redirection service needs once the entire service is up and running. This initial configuration may show to be inappropriate and the locality may have to be improved by creating or removing domains.

4.2 The redirection server

A redirection server has to perform two kinds of tasks: answer incoming requests and manage the location information for the replicas to be found in its domain. Each redirection server can receive requests either from client browsers or from other redirection servers, which we call *client redirection servers*. These requests are known as *lookups*. They do not modify the information stored in the redirection service but allow clients to retrieve addresses of replicas of Web pages. To let the Web servers hosting replicas add and maintain replica information, there is also a need for *update* requests. An update corresponds to either an *insert*, which lets the address of a replica be stored by the redirection service or to a *delete*, which removes an address from the redirection service. Each redirection server has to handle these three types of requests.

The technique for handling requests in the redirection service is the same for both updates and lookups (for more details, see [1, 11]). Requests are always initiated at a lowest-level redirection server. In the case of update requests, the request is forwarded only upwards. For an insert request, the upper domain has to be contacted to ask permission to store an address. If the permission is granted, the upper domain installs a pointer towards its child domain. This happens recursively up until a server is reached that already holds a record for the considered Web page. In such a case there is no need for forwarding the request any further. It simply means that the upper level already has a forwarding pointer installed. Installing forwarding pointers guarantees that any inserted address can be found following a path of pointers from the root to the server where the address is actually stored.

In the case of a delete request, the upper domain has to be contacted only if the record for the Web page at the cur-

rent redirection server becomes empty. In such a case, the pointer at the next higher-level server has to be deleted. This happens recursively up to the root redirection server if necessary. This mechanism guarantees that following a path of forwarding pointers always leads to an address and never to an empty record.

Finally, in the case of a lookup, the request is forwarded upwards in the hierarchy until information is found for the Web page that is being looked up. When a pointer is found, the lookup request is further forwarded downwards to the redirection server referred to by the pointer. The presence of a pointer guarantees us to find an address in one of the subdomains. This address is eventually sent back to the requesting redirection server.

Special attention is necessary when forwarding the requests. Since the information concerning a Web page is available only at one redirection server in each domain, a client redirection server has to know exactly to which other server it should send a request. The redirection server selection is achieved differently depending on the direction the forwarding follows (upwards or downwards). If the request goes upwards, we select the redirection server from the upper domain that is the closest to the home location of the Web page. If the request goes downwards, we simply make use of the forwarding pointer and contact the referenced redirection server.

Each redirection server acts independently when dealing with its contents or the requests it receives. It makes use of local information as much as possible in order to reduce the communication overhead. However, during an update or a lookup request, another redirection server may have to be contacted. If this server is unreachable, the redirection service can make use of a simple fall-back mechanism. Whenever a lookup request takes too long to proceed, the initiating server can take the decision of contacting directly the home server of the document. This prevents a client from indefinitely waiting for a redirection server that is currently unavailable. In the case of update requests, the client does not have to wait until the full completion of the request but can get an answer directly after the update has been completed locally.

In addition to the above tasks, a redirection server has to handle the registration of Web servers willing to participate in the service. This encompasses registering replicas of Web pages and offering space for hosting replicas.

Address caching mechanisms can be applied to distributed redirection. The time-to-live of each address in the cache can be provided directly by the Web server hosting the replica since it has to fulfill a contract determining precisely what it should store and how long. This time value can be given to the redirection server where the address of the replica is stored and be further used as time-to-live value in the cache of the client redirection server. The standard lookup procedure can therefore be short-cut by using the address cache and it can be guaranteed than an address found in the cache is always valid. Note that the contents of a replicated Web page can still change without invalidating the cached addresses.

5 Interacting with the redirection server

The client browser interacts with the redirection server in order to look for replicas of Web pages. This interaction has to occur as transparently as possible. First, a client should not be aware it is dealing with a replica of the Web page it requested. At no point a client should be able to keep an explicit reference towards a replica of a Web page, for example by bookmarking it. Not preventing this can, first, lead to dangling pointers when the replica is removed. This is unacceptable if the original page is itself still accessible. Second, discovering new replicas closer to the client would require an explicit action from the end user, which is also unsatisfying. This is what we call replication transparency.

It is the task of the client's redirection server to maintain replication transparency. This is achieved by not displaying the addresses of the replicas to the client browser. The client sees only the home location of a Web page. It has no way of discovering the address of a replica when using the redirection service and of accessing the replica directly. The redirection server at the client side thus takes care of fetching the replica for the client and acts as if it were the home Web server.

Second, a client browser should not be aware its requests are going through a redirection service. The end user should have the least possible to do to take benefit of the redirection service. This makes the deployment and the use of the redirection service at client sites easier. We can achieve this by carefully integrating the components of the redirection service with their environment. This is what we call transparency of use.

Let us see how an end-user request for loading the http://www.cs.vu.nl/globe/ page is handled when using the redirection service. We assume no Web proxy is installed in the client browser configuration. For loading the page, the browser performs two tasks. First, it resolves the DNS-domain name www.cs.vu.nl into the IP address of a Web server by contacting its authoritative DNS server. Second, it generates an HTTP request for the page and sends it to the Web server whose address was returned. Since redirection has to take place locally at the client side, the redirection service has to integrate between these two steps. By using a modified authoritative DNS server at the client site, we can resolve the client browser's DNS request into the address of its local redirection server (see Figure 2, message exchange 1). Without noticing it, the client browser is therefore asked to contact the redirection service which it

believes to be the Web server of the requested page. This approach realizes transparency of use. As for the second step, the browser sends its HTTP request to its local redirection server (message 2) which looks for a replica (message exchange 3). The lookup can recursively lead to several message exchanges with other redirection servers (not shown in the figure). The client's local redirection server is in charge of fetching the requested Web page (message exchange 4) and returning it to the client (message 5). Using an HTTP-redirect at this stage would violate replication transparency. Note that Web proxies display a similar behavior. Moreover, the HTTP reply the redirection server sends back to the browser has to use the original URL to designate the page and thus preserve replication transparency. It also means that the URLs contained in a replicated Web page are not rewritten to match the location of the replica. Any subsequent request goes through the redirection server and is not bound to a given replica.

The client site's servers that are needed for the redirection service can be integrated into a single component. The modified DNS server may just be a front-end to the redirection server. The redirection server component has therefore to act as (1) a DNS server, resolving DNS names; (2) a Web proxy, receiving client browser requests and fetching pages from Web servers; and (3) a redirection server, performing lookups in the redirection service.

In our example, the Web page http://www.cs.vu.nl/globe/ was known to the redirection service. It may, however, happen that no replica of the requested Web page is found by the redirection service. The page's Web server may not participate in the redirection service or this specific page may not be replicated while others from the same Web server are. The protocol we follow if the requested site is not participating differs slightly from the previous case. Assume that no replica address has been found at the end of the message exchange 3. The redirection server therefore has to contact the home location of the requested Web page since it is the only known Web server for the page. It acts as a Web proxy would do: it resolves the domain name of the requested page and fetches the page from the home Web server. Finally, the redirection server returns the Web page to the browser as it would normally do.

A number of optimizations can be applied to the above protocol. First, when the DNS, proxy and redirection servers are integrated into one single entity, the client browser exchanges two sets of messages with the same entity but using different protocols. We can improve efficiency by relaxing the transparency constraint: we can let the client browser consider the redirection server as a Web proxy. The browser proxy configuration has thus to be adjusted accordingly. The browser has only to send HTTP requests to the redirection server which performs the subsequent lookup. To preserve transparency, the redirection service could be directly integrated at the router or switch level, as it is the case for transparent caches. It would then intercept TCP traffic for port 80.

A second optimization concerns the lookup of a replica address in the redirection service. Consider the case where no participating site is known for the page. A redirection server can only discover this fact after the lookup has been completely processed, that is, when the hierarchy has been traversed without finding any replica address. Since each unsuccessful lookup is costly, we would like to prevent them as much as possible. As an optimization, a redirection server can keep a database of participating Web sites. The database is checked for each client HTTP request before the address lookup in the redirection service starts. The lookup is short-cut if the requested site is not participating, at which point the redirection server performs the name resolution by directly forwarding the request to the local DNS server. The rest of the protocol remains unchanged.

6 Integration with existing platforms

To facilitate installation and deployment, we implement our redirection server as a module in the Apache server. Apache is a well-known and widely-used Web server. It has the advantage of being implemented in a modular way and as such is extendable with new features. Moreover, similar experiments with integrating replication [6] or DNS-based redirection [8] features in Apache modules have shown the approach to be feasible.

Another integration point to be careful with is the naming scheme. To find addresses of replicas of Web pages, the redirection service needs a unique, location-independent identifier for each Web page. An identifier is associated to a number of location-dependent addresses that denote the physical location of the replicas. The de facto naming scheme that is use today (i.e., URLs) does not comply with these requirements. We propose to use URLs as both human-friendly names and addresses of replicas. The redirection service is in charge of translating these names into identifiers and then use the identifiers to look for replica addresses. An identifier is directly derived from a URL. It is composed of a hash and of the geographical coordinates of the home Web server. The coordinates allow a client redirection server to select the redirection server that is closest to the home location of the requested page during lookups. As explained in [10], this approach does not violate the location-independence property since the coordinates of the home location will never be changed.

7 Conclusion

The redirection mechanisms used in today's World-Wide Web such as HTTP-based redirection, DNS-based redirec-

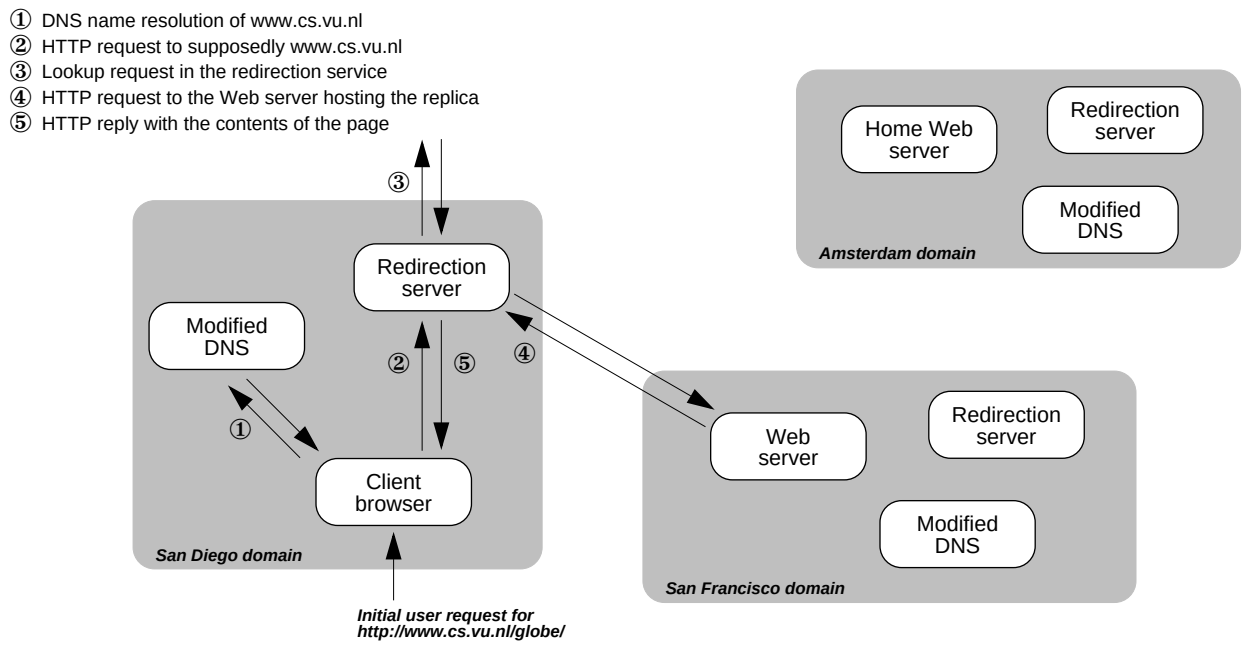

Figure 2. Interaction of the distributed redirection components

tion or TCP handoff exhibit characteristics that make them not fully satisfactory. The main concern is that with any of these methods, a *home-based* approach is used. The request of a client is in most cases redirected only after it has reached the Web page home location. Not only does this put a load on the home Web server and does it generate traffic on the network, but it induces latency that can be perceived by the end user.

We devised a scheme where the redirection is fully distributed. An important aspect is that the locality has to be preserved: the redirection decision has to take place as locally to the client as possible and the selected replica of the requested Web page has to remain close to the client. In such a way, we avoid unnecessary communication for both finding a replica and contacting it. Latency is kept as low as possible.

Distributed redirection makes use of a world-wide collection of redirection servers organized as a collection of trees, one per Web page or group of Web pages from the same leaf domain. Leaf servers store addresses of replicas and perform lookup requests on behalf of clients. A redirection server supports both DNS and HTTP protocols for interacting with clients, as well as its own protocol for looking up and updating addresses of replicas. Each participating client or server site has to run its redirection server.

The distributed redirection mechanisms have to integrate seamlessly in the current World-Wide Web. A redirection server is implemented as an Apache module and used transparently by being configured as an authoritative DNS server. We are currently in the process of running simulations and comparing both DNS-based and distributed redirection. Future work encompasses experiments and performance measurements of our redirection scheme as well as the addition of optimizations such as address caching mechanisms.

References

[1] A. Baggio, G. Ballintijn, M. van Steen, and A. S. Tanenbaum. Efficient tracking of mobile objects in Globe. *The Computer Journal*, 44(5):340–353, 2001.

[2] B. Cain, A. Barbir, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, and O. Spatscheck. Known cn request-routing mechanisms. Internet draft, May 2002.

[3] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A Hierarchical Internet Object Cache. In *Annual Technical Conference*, pages 153–163, San Diego, CA, Jan. 1996. USENIX.

[4] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[5] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Re-

quest Distribution in Cluster-Based Network Servers. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, CA, Oct. 1998. ACM.

[6] G. Pierre and M. van Steen. Globule: a platform for self-replicating web documents. In *6th Int. Conf. on Protocols for Multimedia Systems*, pages 1–11, Enschede, The Netherlands, Oct. 2001.

[7] M. Rabinovich and O. Spastscheck. *Web Caching and Replication*. Addison-Wesley, Reading, MA., 2002.

[8] M. Szymaniak. DNS-based client redirector for the Apache HTTP server. Master's thesis, Warsaw University and Vrije Universiteit, June 2002.

[9] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *19th International Conference on Distributed Computing Systems*, pages 273–284, Austin, TX, June 1999. IEEE.

[10] M. van Steen and G. Ballintijn. Achieving scalability in hierarchical location services. In *26th International Computer Software and Applications Conference (CompSac)*, pages 899–905, Oxford UK, Aug. 2002.

[11] M. van Steen, F. Hauck, G. Ballintijn, and A. Tanenbaum. Algorithmic Design of the Globe Wide-Area Location Service. *The Computer Journal*, 41(5):297–310, 1998.

[12] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Seventeenth Symposium on Operating Systems Principles*, pages 16–31, Kiawah Island Resort, SC, USA, Dec. 1999.